LLNL-CONF-687015

Evaluating the Feasibility of Storage Class Memory as Main Memory

G. S. Lloyd, M. B. Gokhale

March 29, 2016

MEMSYS 2016
Washington DC, DC, United States
October 3, 2016 through October 6, 2016

Evaluating the feasibility of storage class memory as main memory

Scott Lloyd
Lawrence Livermore National Laboratory,
Livermore, CA
lloyd23@llnl.gov

Maya Gokhale
Lawrence Livermore National Laboratory,
Livermore, CA
gokhale2@llnl.gov

ABSTRACT

Storage class memory offers the prospect of large capacity persistent memory with DRAM-like access latency. In this work, we evaluate the performance of a small set of benchmarks using SCM as main memory. We use an FPGA emulator to model a range of memory latencies spanning DRAM to latency projected for SCM and beyond. Our work highlights the performance impact of higher latency and identifies conditions by which SCM can effectively be used as main memory.

CCS Concepts

•**Computing methodologies** → **Modeling and simulation;** •**Computer systems organization** → *Architectures;* •**Hardware** → *Analysis and design of emerging devices and systems; Memory and dense storage;*

Keywords

accelerator; data intensive; emulator; energy; memory bandwidth; performance; persistent memory; processing in memory; storage class memory

1. INTRODUCTION

After many decades of research, persistent memory with DRAM-like access width and latency is appearing in the commercial market. Such persistent memory has been named Storage Class Memory (SCM) to recognize its dual role as a first level storage tier and as a large capacity main memory. There are multiple SCM technologies such as Phase Change Memory, STT-MRAM, and resistive RAMs. The announcement of Intel/Micron's 3D XPoint [1] once again renews interest in the prospect of large (tens of TB) storage class persistent memory local to a compute server.

Figure 1, adapted from [10], shows the memory/storage latency spectrum, with SCM in the 200+ ns latency range. Near DRAM has high bandwidth, low latency, and low capacity, followed by standard DDR3 DRAM. Far DRAM is characterized by memories such as the Hybrid Memory Cube, with very high bandwidth, moderate capacity, and higher latency. The SCM tier has much higher capacity, with latency in the range of 200–2000 ns. SSDs with NAND Flash have multi-TB capacity but their block level access and higher latency makes them unsuitable as SCM.

In this work, we investigate the use of SCM as a CPU's main memory. Such an architecture is emerging with SCM as a likely candidate for an exascale node's capacity memory. At the exascale, power and cost considerations will require a much reduced DRAM-to-processor ratio compared to petascale machines, and system architects look to SCM to provide the additional capacity at very low idle power and relatively low latency. From the data analysis perspective, node local large capacity memory is also highly desirable to store larger partitions of massive data sets local to the server than is possible in DRAM.

To estimate the performance of applications using such a memory, we have measured performance of a small collection of benchmark kernels using an FPGA emulator. The emulator is programmed to deliver memory access latency projected for various memory technologies from very fast DRAM to SCM. Our study sweeps a range of latencies for each benchmark to assess to what extent storage class memory can serve directly as a main memory with existing cache hierarchy. We vary the read/write latency ratio to study the effects of longer write latency on application speed. Finally, we incorporate application-specific caching in the memory (active memory) to assess its benefit in SCM.

2. EVALUATION METHODOLOGY

Our evaluation uses an FPGA emulator (Figure 2) that records memory transactions generated by software applications without affecting cache, memory traffic, or emulated speed of the application. The emulator runs on a Xilinx ZC706 development board. The board has a Zynq 7045 System-on-Chip with a dual ARM CPU plus programmable logic, two on-board DDR3 memories, and other I/Os. The ARM processor cores have separate L1 (32 KB, 4-way set-associative instruction and data) and shared L2 (512 KB, 8-way set-associative) caches. A 1 GB 32-bit wide DDR3

Size (GB)	0.1-8	8–64	8–64	64–128
Latency (ns)	45	70	100	200+
Type	Near	DDR3	Far	SCM

Figure 1: Memory/storage hierarchy

MEMSYS '16, October 03-06, 2016, Alexandria, VA, USA
© 2016 ACM. ISBN 978-1-4503-4305-3/16/10. . . $15.00
DOI: http://dx.doi.org/10.1145/2989081.2989118

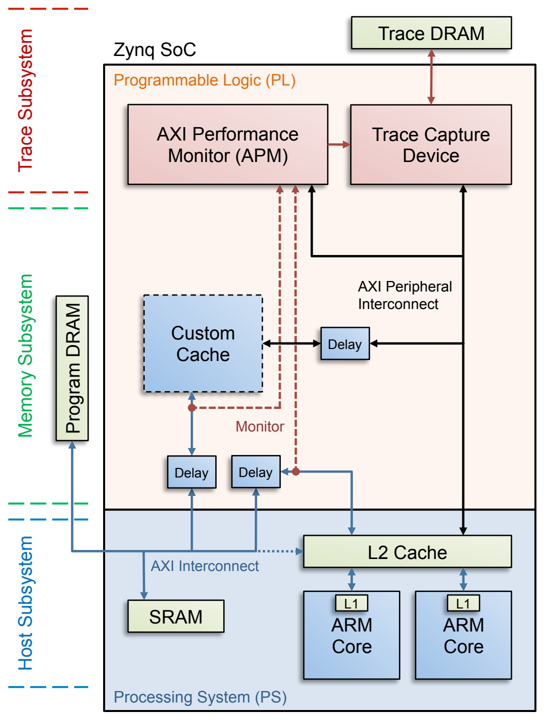

Figure 2: Zynq SoC with emulation framework

memory is used to hold application code and data.

The ARM cores can run at a clock frequency ranging from 1 MHz to 800 MHz. For these experiments, the ARM runs at 128.6 MHz to achieve the emulated CPU frequency of 2.57 GHz. We have implemented programmable delay units in the FPGA fabric to emulate different memory latencies. The delay units are programmable at a range of 0–262 us in 0.25 ns increments. The emulated memory subsystem has multiple memory channels and is capable of accepting up to 16 concurrent memory requests and multiple access widths. We have made our emulator available as open source [2].

We additionally have added logic in the FPGA to emulate an application-specific cache. The Data Rearrangement Engine (DRE) [6] performs in-memory gather/scatter and strided DMA operations using an SRAM scratchpad. The application-specific cache is shown as a dotted box in the programming logic part of Figure 2[1]. While the main CPU accesses memory at cache line (32 byte) granularity, the memory-side DRE can access memory at both cache line and narrow 8 byte granularity. The custom cache is invoked explicitly from the CPU and can operate concurrently with the CPU.

Statistics collected by the emulator include emulated run time and traced memory read and write requests of both the serial program running on a single ARM core and the in-memory cache, memory access counts, and total bytes transferred. In previous work, we have merged multiple serial traces to emulate a data parallel workload [7].

3. EXPERIMENTS

Using the emulator, we have evaluated how SCM characteristics affect the performance of a small set of benchmarks. We vary latency from near memory through several projected SCM values, specifically, latencies of 45 ns, 70 ns, 100 ns, 200 ns, 400 ns, 800 ns, 2 us, 4 us, and 8 us (extending beyond SCM tier). We approximate the effects of working set size relative to processor cache size by running different problem (and therefore dataset) sizes, thereby varying the cache-to-memory (CM) ratio of the benchmark. As some SCMs have slower write latency than read, the read/write latency ratio is also varied, at ratios of 1:1, 1:2, 1:4, and 1:8. Finally we introduce the application-specific cache that performs custom DMA and gather/scatter operations and communicates blocks of reduced data to/from the main CPU. All graphs show latency on the x-axis and runtime on the y-axis, meaning lower is better.

Benchmarks are as follows:

- DGEMM, dense matrix multiply that is part of the HPC Challenge benchmark suite [4]. DGEMM characterizes compute-intensive, cache-friendly kernels.
- Random Access, to measure the rate of random integer updates. This “gups” benchmark, also part of the HPC Challenge set, stresses the memory system with random read-modify-write sequences.
- ImageDiff, an in-house benchmark that computes the pixel-wise difference of two images, using 16X reduced resolution (i.e. access every 16th pixel in each image). This benchmark is modeled after image processing tasks on high resolution imagery that sub-sample and analyze the reduced size image.
- STREAM [9], the open source benchmark that “measures sustainable memory bandwidth (in MB/s) and the corresponding computation rate for simple vector kernels.” STREAM is included in the HPC Challenge set.
- SparseMatVec, a sparse matrix multiply with a dense vector, using an open source implementation [5].
- PageRank, in-house implementation of the popular algorithm to weight importance of web pages.
- Breadth First Search (BFS), in-house implementation of the first Graph500 benchmark [3].
- Kmer, an in-house benchmark for a key/value store. The keys are k-length genome fragments and the values are a list of genomes and their positions that contain the k-length fragments. Two K/V store implementations are evaluated, a btree and a hash table.

3.1 Cache-to-memory (CM) ratio

The first experiment measures DGEMM elapsed time at nine latency levels and seven CM ratios. The sizes of the matrices were adjusted to adhere to the CM ratio. The curves in Figure 3 showing the runtime profile of DGEMM indicate that for latency up to 800 ns, the size of the working set can be double the cache size with little impact on performance. At CM ratios of 1:4 and higher, runtime increases linearly with latency. This result suggests that dense matrix multiply kernels for small matrices can tolerate the lower range of SCM latencies. Large dense matrices in SCM can be decomposed into blocks, e.g. machine-learning weight-matrix

[1]and is not included in the open source software.

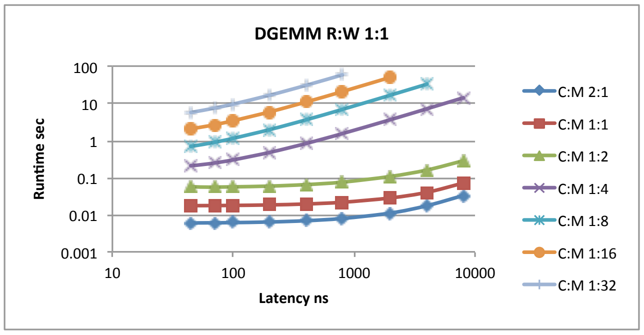

Figure 3: DGEMM with 1:1 read/write ratio at varying latencies and varying cache-to-memory ratios

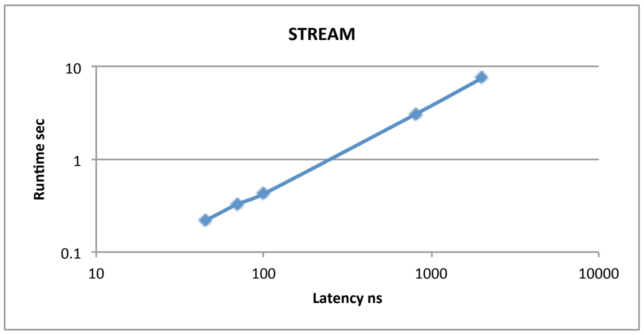

Figure 4: STREAM at varying latencies

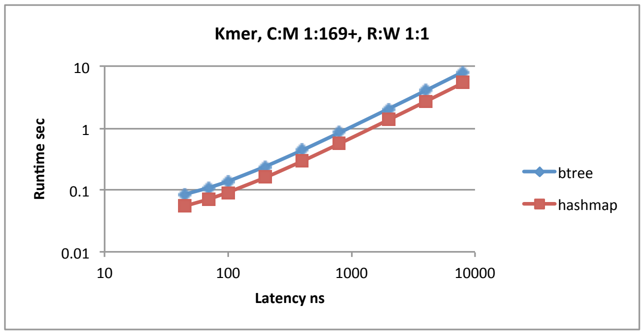

Figure 5: Kmer benchmark at varying latencies, two index schemes

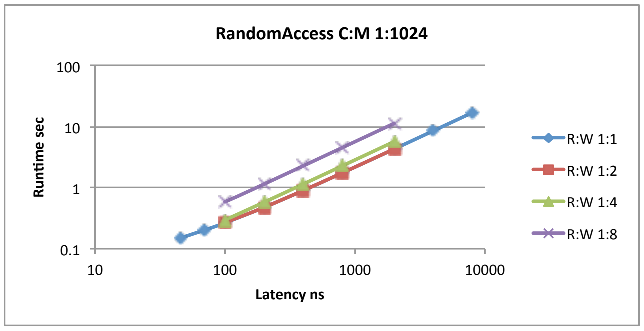

Figure 6: Random Access at varying latencies and r/w ratios

updates. This would allow large neural network weight matrices to reside in the SCM, with small windows – up to twice the cache capacity – being successively updated in an iteration step.

In contrast, the STREAM benchmark (Figure 4) represents workloads with large, uniform data streams that are not suitable for block decomposition. The performance curve shows a direct linear correlation between memory latency and runtime, indicating that using SCM with higher latency results in slower application execution. The other benchmarks show a similar linear correlation. The kmer key/value benchmark profile is shown in Figure 5 for two indexing methods, a btree, suitable for block access, and hash table, a high-performance in-memory index implementation. Kmer also shows linear slowdown with increasing latency. With these latency sensitive workloads, it is desirable to run multiple data parallel threads, transforming the computation to be throughput driven. Concurrent worker threads utilizing available memory bandwidth could in aggregate compensate for longer SCM latency, and thus take advantage of SCM capacity (see [7] for a detailed study of throughput driven applications on the Hybrid Memory Cube).

3.2 Read/Write Asymmetry

Next we consider the effect of read/write ratio on performance. Figure 6 and Figure 7 show runtime at varying latencies for four read/write ratios for the RandomAccess and STREAM benchmarks. The graphs show that an asymmetric read-write ratio up to 1:4 has little impact on performance, indicating that cache can mask the latency of the write. This result holds for opposite ends of the spectrum, from completely random access to sequential streams. Since both benchmarks update data in-place in the cache, the read/write ratio is not a factor in performance until 1:8. This result indicates that even without a write buffer in the SCM subsystem, CPU cache can hide long write latency in the presence of write-heavy workloads. Other benchmarks (BFS, ImageDiff, PageRank, SpMV) show no change in performance even at a 1:8 ratio. These benchmarks are read-dominated, so that the increased write latency does not have measurable effect.

3.3 Application-specific cache using Data Rearrangement Engine

The final set of experiments incorporate an application-

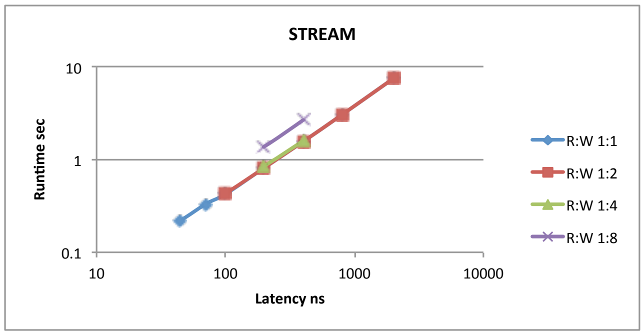

Figure 7: STREAM at varying latencies and r/w ratios

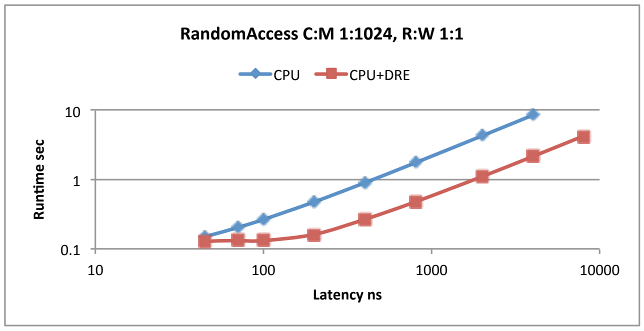

Figure 8: RandomAccess using custom scratchpad at varying latencies ratio

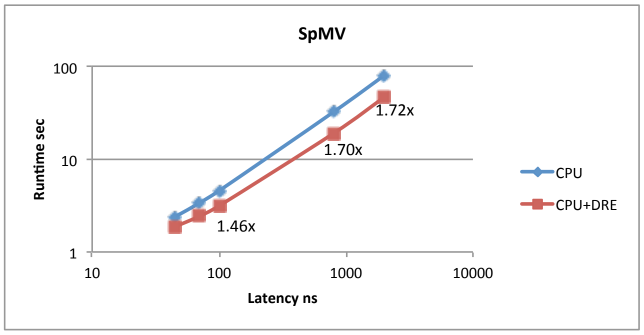

Figure 9: SparseMatVec using custom scratchpad at varying latencies ratio

specific cache, the Data Rearrangement Engine (DRE)[8] in the memory. The DRE is invoked by explicit application command to do a strided DMA or gather/scatter operation between memory and in-memory scratchpad. Figures 8 and 9 show a substantial performance improvement by using a DRE. In the Figures, "CPU" uses the existing CPU cache only. "CPU+DRE" uses an in-memory Data Rearrangement Engine configured to load an SRAM scratchpad with selected bytes using strided DMA. The actual computation is still performed in the CPU. In these algorithm variants, the CPU issues a command to the DRE to gather or scatter a block of data to/from the in-memory scratchpad, then waits for DRE completion before advancing to the next step of the algorithm. Even with this serialization, there is significant performance improvement by using the DRE. The performance improvement is due to the memory model of 16 parallel channels and narrow access width, both of which the DRE can exploit on behalf of the CPU thread.

Figure 10 compares performance of three versions of the ImageDiff benchmark. In addition to "CPU" and "CPU+DRE" which operate as described above, a third version "CPU+DRE w/overlap" overlaps CPU and DRE execution, allowing the CPU to compute image difference on one block while the DRE assembles the next. At latency up to 800 ns, the overlap results in additional performance improvement beyond "CPU+DRE." At the highest latencies, the overlap benefit is overshadowed by the high access latency.

4. DISCUSSION

To summarize the key findings:

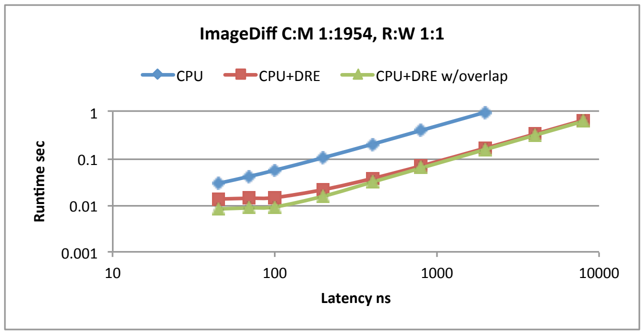

Figure 10: Image differencing of reduced resolution imagery using custom scratchpad at varying latencies ratio

- Caches can do well at hiding latency of storage class memory as long as the working data set size is within 2X of the cache size (Figure 3).
- Applications that have an unbounded dataset size not suitable for cache-friendly blocking slow down linearly with latency (Figures 4, 5). For these applications, a throughput driven approach is desirable to hide latency with concurrent data parallel memory accesses.
- Caches can hide write latencies that are longer than read latencies up to a factor of about 4 for applications with significant write access (Figures 6 and 7) and higher latency for read-dominated workloads.
- A specialized cache (Figures 8, 9, 10) can improve performance significantly at all latencies. Overlapping CPU with the DRE cache gives additional performance improvement up to 800 ns latency.

Many research issues remain to be explored. This study has used a simple cache hierarchy of SRAM caches and SCM and studied benchmarks that either can exploit the processor cache (DGEMM) or have data sets and access patterns that are unlikely to benefit from cache. However, performance of the application benchmarks using a more complex hierarchy that includes DRAM cache/scratchpads [11] should also be investigated.

This study has used a simple abstract model of the memory system in terms of latency, access width, and concurrency. More detailed models, including those specific to a particular SCM technology, can be assessed through an iterative process when evaluation hardware or a low level cycle accurate SCM device simulator is available. First, traces are captured by the emulator for a benchmark run using estimates for the latency. Then, traces are replayed on hardware or a detailed simulator that is instrumented to acquire latency statistics. The average latency obtained from the replayed traces is then entered into the emulator for another run. These steps are repeated a few times until the average latency from trace replay converges to a stable number. After convergence, benchmarks run on the emulator will report more accurate run times for the traced region. An example of this iterative process is given in [7] with a Hybrid Memory Cube board instrumented to report latency dynamically as the trace is replayed.

This study has focused on speed performance. However, energy is as important a factor. SCM power models must be included in the analysis to understand the energy usage for SCM at the workload level and determine whether projections of SCM energy benefit are borne out when used as main memory.

The FPGA emulator has allowed us to measure performance very accurately and run a large set of studies quickly compared to software simulation. However, the emulator has a fixed framework of CPU and cache hierarchy, and more comprehensive studies with other processor/cache models will also be valuable.

5. ACKNOWLEDGMENTS

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract No. DE-AC52-07NA27344. This work was supported by Lawrence Livermore National Laboratory LDRD project 16-ERD-005. LLNL-CONF-687015

6. REFERENCES

[1] 3D XPoint. https://en.wikipedia.org/wiki/3D_XPoint.

[2] Emulator Software. https://bitbucket.org/PerMA/emulator_st.

[3] Graph500. www.graph500.org.

[4] HPC Challenge Benchmark. http://icl.cs.utk.edu/hpcc/.

[5] H. Gahvari, M. Hoemmen, J. Demmel, and K. Yelick. Benchmarking sparse matrix-vector multiply in five minutes. In *SPEC Benchmark Workshop*, January 2007.

[6] M. Gokhale, S. Lloyd, and C. Hajas. Near memory data structure rearrangement. In *Proceedings of the 2015 International Symposium on Memory Systems*, MEMSYS '15, pages 283–290, Washington DC, USA, Oct 2015. ACM.

[7] M. Gokhale, S. Lloyd, and C. Macaraeg. Hybrid Memory Cube performance characterization on data-centric workloads. In *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*, IA3 '15, pages 7:1–7:8, Austin, TX, USA, Nov 2015. ACM.

[8] S. Lloyd and M. Gokhale. In-memory data rearrangement for irregular, data-intensive computing. *IEEE Computer*, 48(8):18–25, Aug 2015.

[9] J. D. McCalpin. STREAM: sustainable memory bandwidth in high performance computers. Technical report, University of Virginia, Charlottesville, Virginia, 1991-2007. A continually updated technical report.

[10] T. Pawlowski. Vision of processor-memory systems (keynote presentation). *International Symposium on Memory Systems*, October 2015.

[11] C. Su, D. Roberts, E. A. León, K. W. Cameron, B. R. de Supinski, G. H. Loh, and D. S. Nikolopoulos. HpMC: an energy-aware management system of multi-level memory architectures. In *Proceedings of the 2015 International Symposium on Memory Systems, MEMSYS 2015, Washington DC, USA, October 5-8*, pages 167–178, 2015.